Crossing boarders with the bus

Integrating enterprise data with public APIs

What is Mule?

Not a donkey

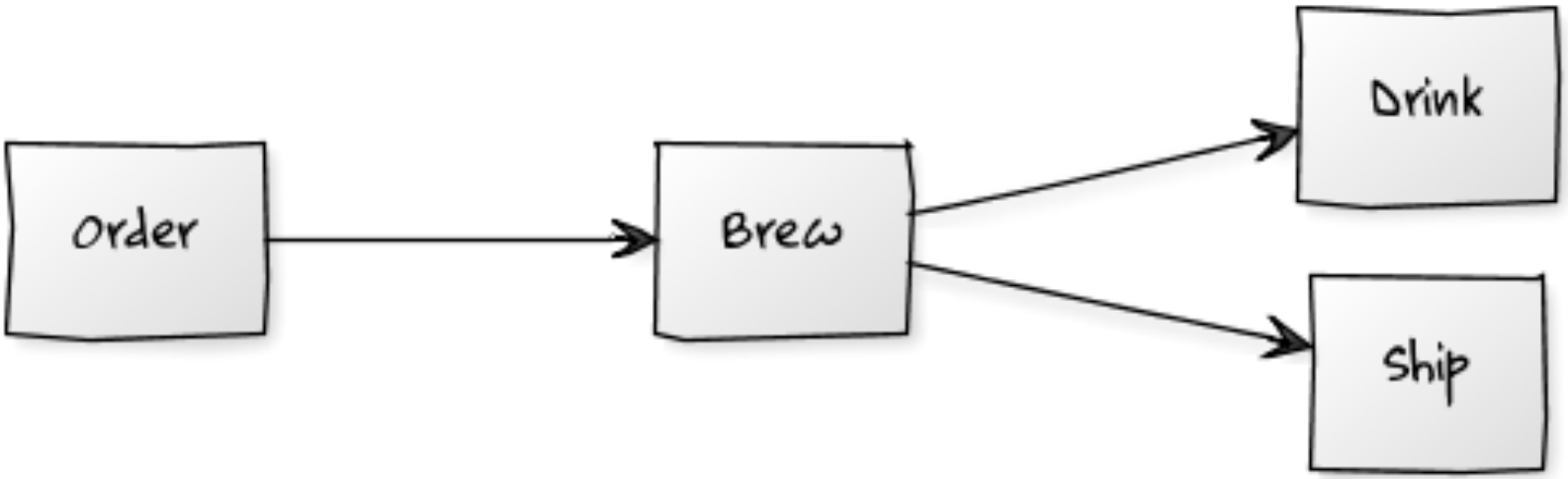
Not a llama

Not a camel

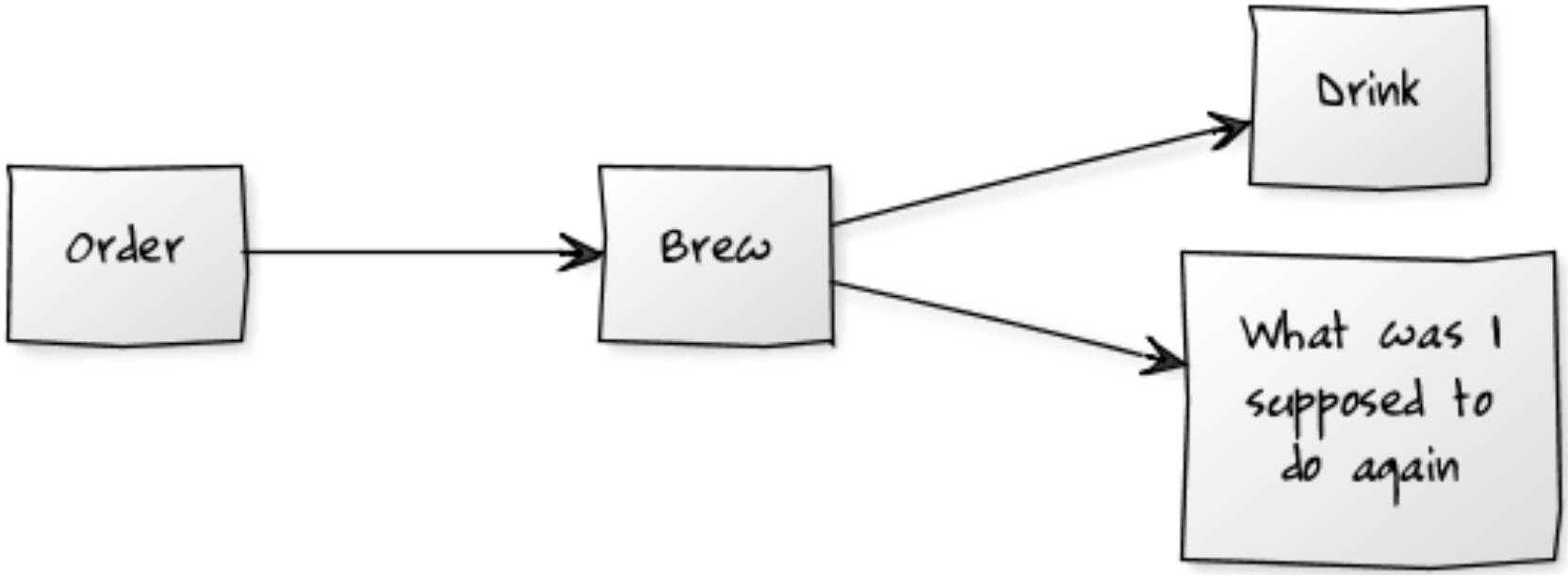
BaaS: Beer As A Service

BaaS

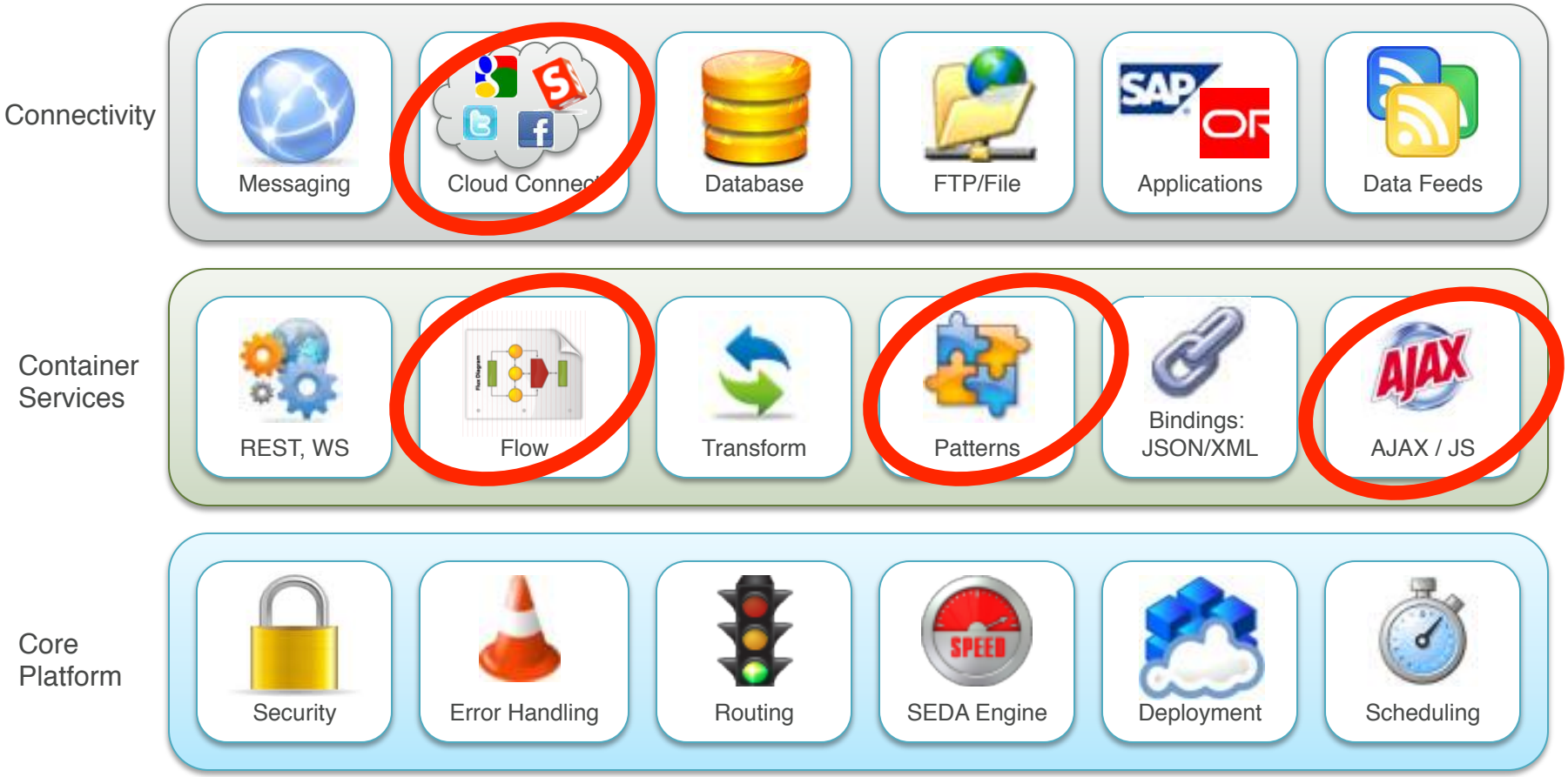
BaaS



BaaS



Mule Stuff


Connectivity
Messaging
Cloud Connect
Database
FTP/File
Applications
Data Feeds
Container Services
REST, WS
Flow
Transform
Patterns
Bindings: JSON/XML
AJAX / JS
Core Platform
Security
Error Handling
Routing
SEDA Engine
Deployment
Scheduling

AJAX – Hook events to the Browser

- Real time events to the Browser
- Trigger ESB events from the Browser
- Simple Set up, no need for App Server
- JSON Support
  - Object Binding
  - Transformers
  - Query Language
- JavaScript client for Mule ESB

AJAX – Hook the ESB to the Browser

Mule Config (Server)

```
<ajax:connector name="ajax" serverUrl="http://localhost:8081/foo”/>

<flow name=”AjaxService">
    <vm:inbound-endpoint path="in"/>

    <component class="org.foo.MyComponent"/>

    <ajax:outbound-endpoint channel="/test"/>
</flow>
```

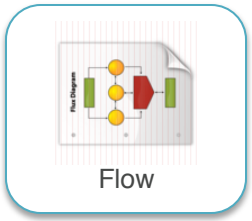
Mule JS Client (Browser)

```
<script type="text/javascript" src="mule-resource/js/mule.js”/>

mule.subscribe("/test", callback);

function callback(coord) {
   //do stuff
}
```

Cloud Connect

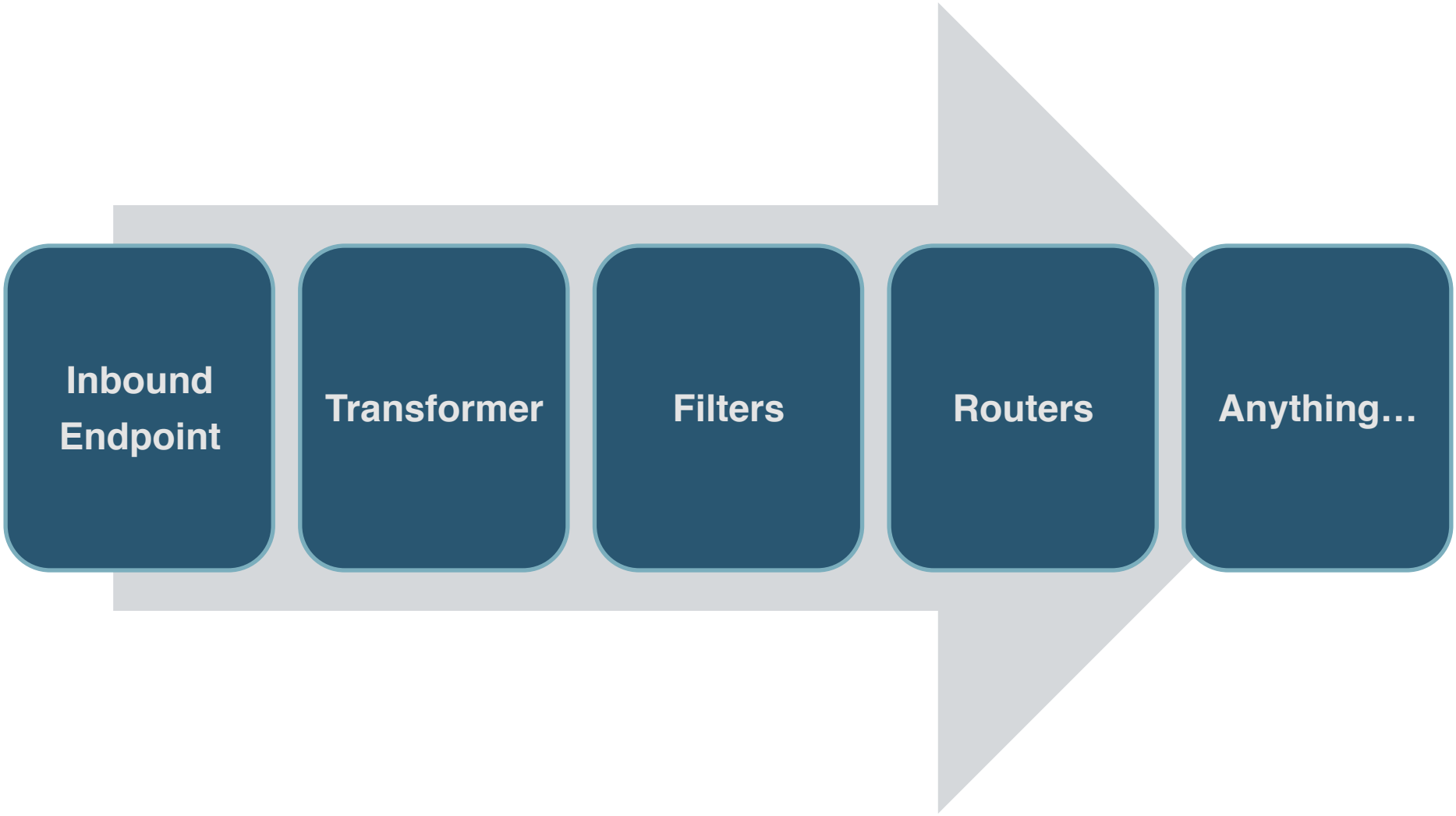
- Integrate with stuff on the net
  - Grab data from SaaS platforms
  - Utilise infrastructure Services
  - Connect to Social Media platforms
- Connectors for cloud applications
- Support for REST and Web Services
- Data bindings using XML and JSON

Mule Flow



- Fluid DSL for creating message flows
- Fixes the ‘S’ in SOA
  - Easier to think about what rather than how
- Mix and match routing, transforms, components
- Much less configuration than Mule 2

<flow>



Demo

<flow> intro

```
<flow name="BeerAsAService">
  <inbound-endpoint address="http://localhost:9090/brew"
                    exchange-pattern="request-response"/>
  <append-string-transformer message="Ordering, "/>
  <append-string-transformer message="Brewing me some beer, "/>
  <append-string-transformer message="Shipping"/>
</flow>

Result: Ordering, Brewing me some beer, Shipping
```

<choice>

```
<flow name="BeerAsAService">
  <inbound-endpoint address="http://localhost:9090/brew"
                    exchange-pattern="request-response"/>
  <choice>
    <when expression="#[groovy:payload == 'IPA']">
      <append-string-transformer
          message="Using recipe with extra hops, "/>
    </when>
    <otherwise>
      <append-string-transformer message="Using normal recipe, "/>
    </otherwise>
  </choice>
…
</flow>
```

<async>

```
<flow name="BeerAsync">
    <inbound-endpoint address="http://localhost:9090/async"
                      exchange-pattern="request-response"/>
    <append-string-transformer message="Ordering, "/>
    <append-string-transformer message="Brewing me some beer, "/>
    <async>
        <expression-transformer>
            <return-argument expression="'Drinking me some beer'"
                             evaluator="string"/>
        </expression-transformer>
        <stdio:outbound-endpoint system="OUT" />
    </async>
    <append-string-transformer message="Shipping"/>
</flow>
```

<all>

```
<flow name="NotifyStaffDrinkers">
  <inbound-endpoint address="http://localhost:9090/all"
exchange-pattern="request-response"/>
  <append-string-transformer message="Ordering, "/>
  <append-string-transformer message="Brewing me some beer, "/>
  <all>
    <outbound-endpoint address="vm://staffDrinker1"/>
    <outbound-endpoint address="vm://staffDrinker2"/>
  </all>
  <append-string-transformer message="Shipping"/>
</flow>
```

Composable, reusable

```
<flow name="CustomMP">
    <inbound-endpoint
       address="http://localhost:9090/custom-mp"
       exchange-pattern="request-response"/>
    <append-string-transformer message="Ordering, "/>
    <flow-ref name="Brewing"/>
    <append-string-transformer message="Shipping"/>
</flow>

<flow name="Brewing">
    <!-- Do brewing -->
    <append-string-transformer message="Brewing me some beer, "/>
</flow>
```

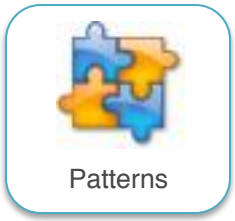
<service> still supported!
But, compare to Mule 2.x...

Loanbroker: Service vs Flow

```
<!--
The loan broker is used to receive loan requests
-->
<service name="TheLoanBroker">
    <inbound>
        <inbound-endpoint ref="CustomerRequests"/>
    </inbound>

    <component class="org.mule.example.loanbroker.esn.SynchronousLoanBroker">
        <binding interface="org.mule.example.loanbroker.credit.CreditAgencyService">
            <outbound-endpoint ref="CreditAgency"/>
        </binding>
    </component>

    <outbound>
        <pass-through-router>
            <outbound-endpoint ref="LenderService"/>
        </pass-through-router>
    </outbound>

    <async-reply timeout="1000000">
        <inbound-endpoint ref="LoanBrokerQuotes"/>
        <custom-async-reply-router class="org.mule.example.loanbroker.routers.BankQuotesResponseAggregator"/>
    </async-reply>
</service>

<!--
The credit agency service will get the credit profile for a customer
-->
<service name="TheCreditAgencyService">
    <inbound>
        <inbound-endpoint ref="CreditAgencyIn"/>
    </inbound>
    <component class="org.mule.example.loanbroker.credit.DefaultCreditAgency"/>
</service>

<!--
The Lender service is used to determine which banks to contact for a quote
-->
<service name="TheLenderService">
    <inbound>
        <inbound-endpoint ref="LenderService"/>
    </inbound>
    <component class="org.mule.example.loanbroker.lender.DefaultLender"/>
    <outbound>
        <pass-through-router>
            <outbound-endpoint ref="BankGateway" transformer-refs="SetLendersAsRecipients"/>
        </pass-through-router>
    </outbound>
</service>

<service name="TheBankGateway">
    <inbound>
        <inbound-endpoint ref="BankGateway"/>
        <forwarding-router/>
    </inbound>
    <outbound>
        <static-recipient-list-router>
            <reply-to address="LoanBrokerQuotes"/>
            <message-property-filter pattern="recipients!=null"/>
        </static-recipient-list-router>
        <logging-catch-all-strategy/>
    </outbound>
</service>
```

```
<flow name="loan-broker">
        <inbound-endpoint ref="CustomerRequests" exchange-pattern="request-response" />
        <component class="org.mule.example.loanbroker.esn.SynchronousLoanBroker">
            <binding interface="org.mule.example.loanbroker.credit.CreditAgencyService">
                <outbound-endpoint ref="CreditAgency" exchange-pattern="request-response" />
            </binding>
        </component>
        <component class="org.mule.example.loanbroker.lender.DefaultLender" />
        <expression-filter expression="payload.lenders.endpoint!=null" evaluator="groovy" />
        <request-reply timeout="10000">
            <recipient-list evaluator="groovy" expression="payload.lenders.endpoint"/>
            <inbound-endpoint ref="LoanBrokerQuotes">
                <custom-aggregator class="org.mule.example.loanbroker.routers.BankQuotesResponseAggregator" />
            </inbound-endpoint>
        </request-reply>
    </flow>
```

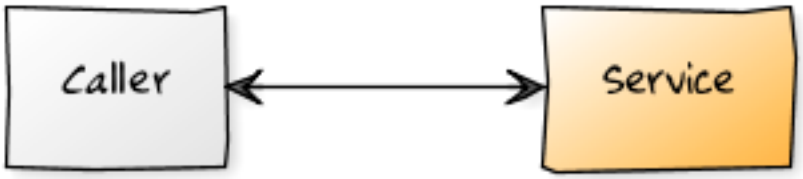
Mule Patterns



- Bigger Building Blocks
  - A level above EIP
- Reduces the amount of Config
- Built in the Core of Mule
  - Users can add their own
  - Can be abstract
- Good for working with external Teams

Patterns – Simple Service

Use Case: Using Mule as a deployment platform for business services.



Configured with JAX-RS annotations

```
<simple-service
        name="weather-service"
        address="http://localhost:6099/weather"
        component-class="org.mule.test.WeatherForecaster"
        type="jax-rs" />
```

Create a JAX-RS service, also supports JAX-WS

Pattern - Web-Service Proxy

Use Case:

- Exposes an external web-service internally, while optionally performing transformations on the SOAP envelope.
- Supports WSDL proxying (with rewriting of the port address component) and overriding (with usage of a provided WSDL).


```
<ws:proxy name="beer-ws-proxy"
    inboundAddress="http://localhost/beer-ws"
    outboundAddress="http://nogne-o.com/beer-ws"
    wsdlFile="localWsdl.wsdl" />
```

Pattern - Web-Service Proxy in Mule 2

```
<spring:bean name="WSProxyService"
    class="org.mule.transport.soap.WSProxyService">
    <spring:property name="wsdlFile" value="localWsdl.wsdl"/>
</spring:bean>

<service name="httpWebServiceProxy">
    <inbound>
          <inbound-endpoint address="http://localhost:8080/my/service"
    synchronous="true"/>
    </inbound>
    <component>
          <spring-object bean="WSProxyService" />
    </component>
    <outbound>
          <pass-through-router>
                    <outbound-endpoint
           address=http://acme.come/remote/web/service
          synchronous="true"/>
          </pass-through-router>
    </outbound>
</service>
```

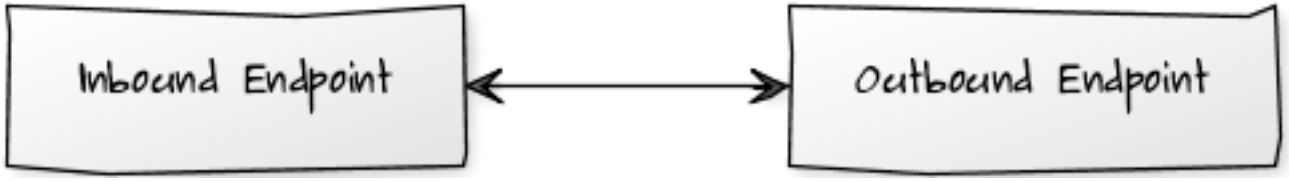
Pattern - Web-Service Proxy

Pattern - Web-Service Proxy

```
<mulexml:xslt-transformer
      name="v1-to-v2"
      xsl-file="v1-to-v2.xsl"/>
<mulexml:xslt-transformer
      name="v2-to-v1"
      xsl-file="v2-to-v1.xsl"/>

<ws:proxy name="beer-ws-proxy"
     inboundAddress="http://localhost/beer-ws"
     outboundAddress="http://nogne-o.com/beer-ws"
     transformer-refs="v2-to-v1"
     responseTransformer-refs="v1-to-v2"
     wsdlFile="beer.wsdl" />
```

Pattern - Bridge

Use Case: Transactional synchronous bridge, for example: JMS bridging (topic to queue)




```
<bridge
        name="transacted-bridge"
        transacted="true"
        inboundAddress="wmq://test.In"
        outboundAddress="jms://test.Out" />
```

REST

- Uses Jersey
  - JAX-RS reference implementation
- Build *correct* RESTful services
- Lightweight, easy to use

REST: Why Jersey in Mule?

- Wrap in a transaction, send off to JMS
- Implement custom routing rules based on URLs
- Mule security
- Easily transform old versions

Service Class

```
@Path("/")
public class BeerOnDemand
{
    @POST
    @Consumes({ "application/json" })
    @Produces({ "application/json" })
    public OrderStatus order(Order order)
    {
        // return a status
        return …;
    }

    @GET
    @Produces({ "application/json" })
    public String brew() {
        return "Brewing!";
    }
}
```

Jersey Resources

```
<flow name="BeerAsARestService">
    <inbound-endpoint address="http://localhost:9090/jersey"/>
    <jersey:resources>
        <component class="com.mulesoft.beer.BeerOnDemand"/>
        <component class="com.mulesoft.beer.CustomerPortal"/>
    </jersey:resources>
</flow>
```

Easy REST client with JSON

```
<flow name="OrderPlacer">
      …
    <json:object-to-json  />
    <response>
        <json:json-to-object
              returnClass="com.mulesoft.beer.Order" />
    </response>
    <outbound-endpoint
        address="http://localhost:9090/rest-beer"/>
</flow>
```

On the wire

Object

```
public class Order {
  private String
  customer;
  private intcases;

}
```

JSON

```
{
  “customer” : “Dan”
  “cases” : 300
}
```

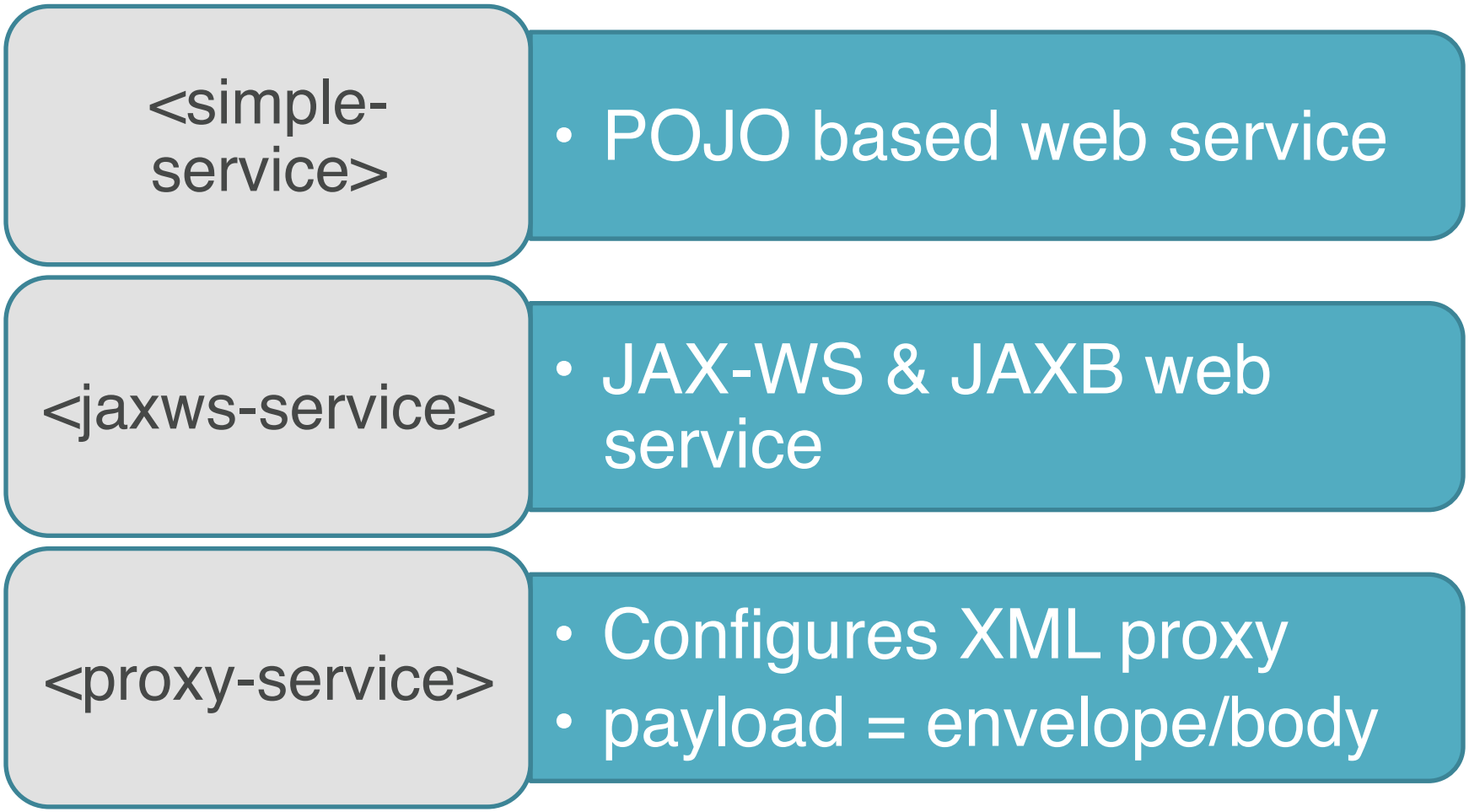
Web Services

- Web Services are a PITA
  - Mule 3 makes them easier
- Host and Consume Web services
- Decoupled Transport from Protocol
  - Receive / Send over JMS, email, etc
- JAX-WS or Simple
- Easy to Proxy and modify

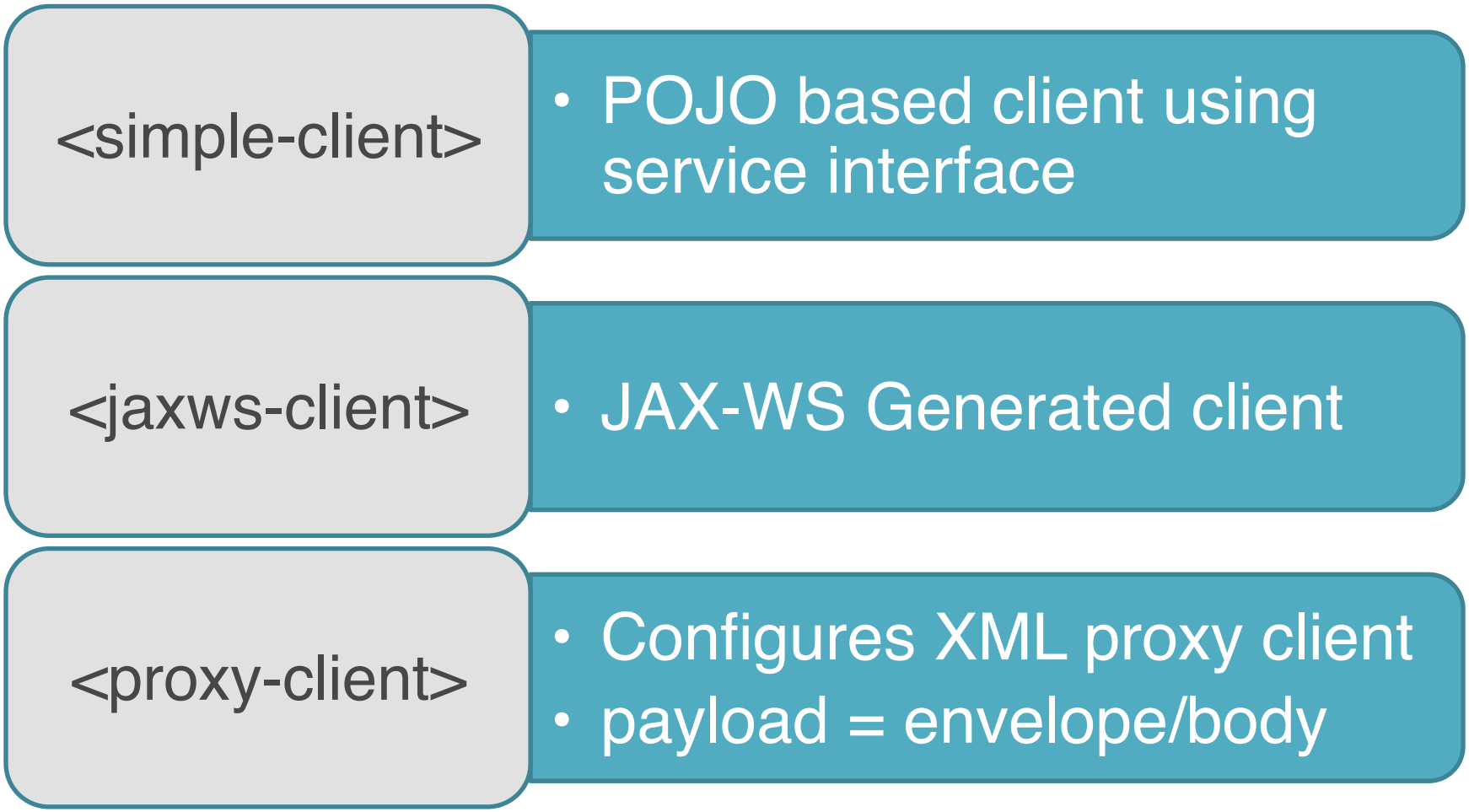
Hosting web services

```
<flow name="BeerAsAWebService">
    <inbound-endpoint address="http://localhost:9090/ws"/>
    <cxf:jaxws-service
        serviceClass="com.mulesoft.beer.BeerOnDemand"/>
    <component class="com.mulesoft.beer.BeerOnDemand"/>
</flow>
```

Web Services: Improved XML configuration



Web Services: Improved XML configuration



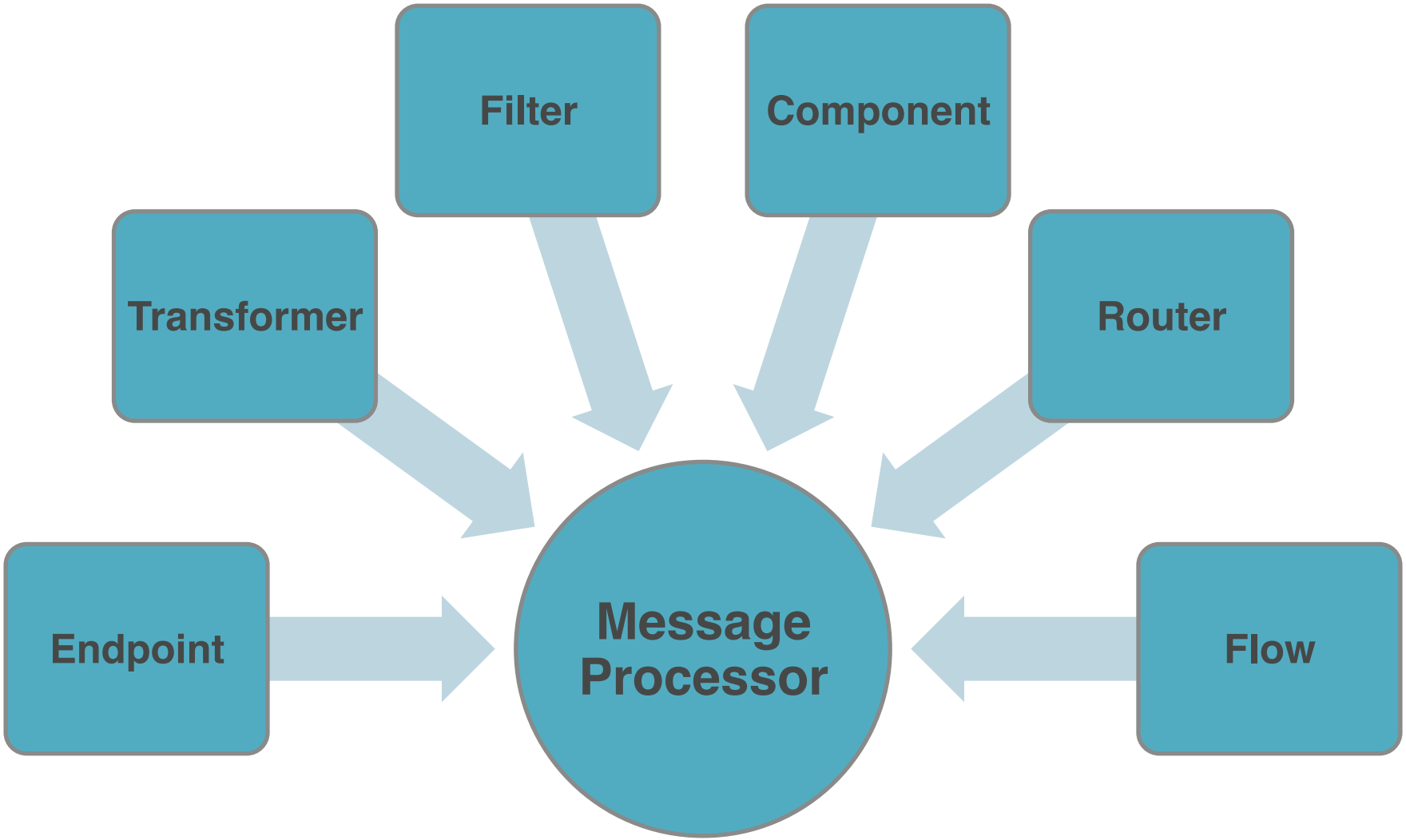
Flexible, Scriptable

```
<flow name="BeerAsAWebService">
  <inbound-endpoint address="http://localhost:9090/ws-async"/>
  <cxf:jaxws-service
        serviceClass="com.mulesoft.beer.BeerOnDemand"/>
  <expression-transformer>
    <return-argument
        expression="'Processing ' + payload[0] + ' orders'"
        evaluator="groovy"/>
  </expression-transformer>
</flow>
```

How does this all work?

MESSAGE PROCESSORS

Message Processors and Flows



Custom MPs

```
<flow name="BeerAsAService">
    <inbound-endpoint address="http://localhost:9090/brew"/>
    <custom-processor
          class="com.mulesoft.beer.BrewerMessageProcessor"/>
</flow>
```

For example….

```
public class BrewerMessageProcessor
    implements MessageProcessor
{
    public MuleEvent process(MuleEvent event)
        throws MuleException
    {
        event.getMessage().setPayload(
            "Yup, fermentin' me some beer.");
        return event;
    }
}
```

For example….

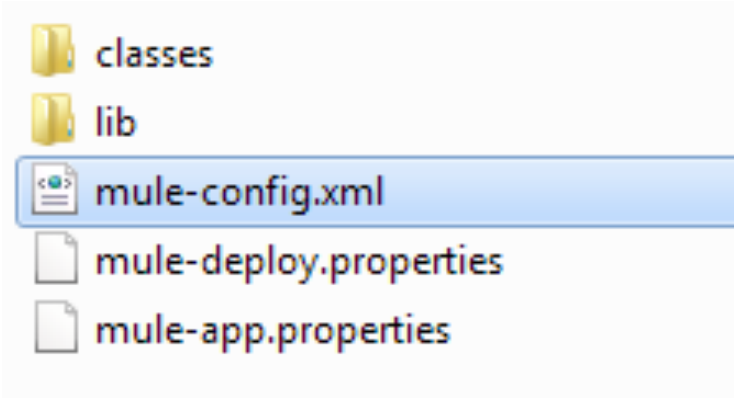
```
public class BrewerMessageProcessor
    extends AbstractInteceptingMessageProcessor
{
    public MuleEvent process(MuleEvent event)
        throws MuleException
    {
        event.getMessage().setPayload(
           "Yup, fermentin' me some beer.");
        MuleEvent response = processNext(event);

        event.getMessage().setPayload(
           "300 cases brewed.");
        return respones;
    }
}
```

Moving in when the time is right

HOT DEPLOYMENT

- Pragmatic
- #NoOSGi
- Not another app server

Applications are zips



- Apps are zips or exploded directories
- By convention: most parts are optional

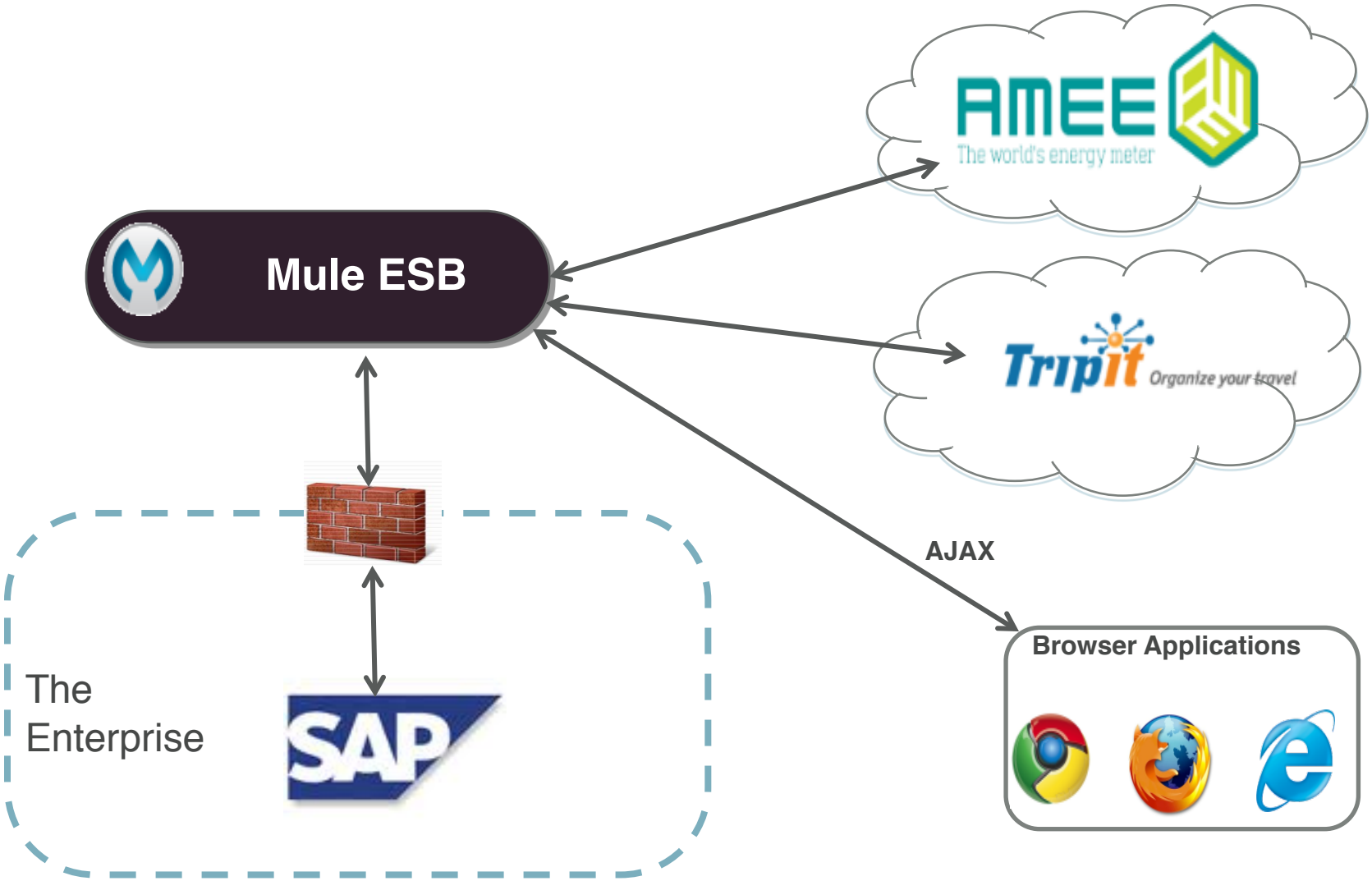
Deployment Descriptor

- domain
- config.resources
- redeployment.enabled
- encoding
- config.builder

Maven Plugin

```
<project>
...
    <packaging>mule</packaging>
...
</project>
```

Enterprise to Cloud Integration



http://mulesoft.org
@rossjmason